Available online at www.sciencedirect.com
ScienceDirect
Procedia Computer Science 1 (2012) 2669–2677
Procedia Computer Science
www.elsevier.com/locate/procedia

International Conference on Computational Science, ICCS 2010

On distributing load in cloud computing: A real application for very-large image datasets

Raúl Alonso-Calvo, Jose Crespo, Miguel García-Remesal, Alberto Anguita and Victor Maojo

GIB – LIA
DLSIIS & DIA
Facultad de Informática
Universidad Politécnica de Madrid
28660 Boadilla del Monte (Madrid), Spain
{ralonso, jcrespo, mgarcia, aanguita, vmaojo}@infomed.dia.fi.upm.es

Abstract

Managing large image collections has become an important issue for information companies and institutions. We present a cloud computing service and its application for the storage and analysis of very-large images. This service has been implemented using multiple distributed and collaborative agents. For image storage and analysis, a region-oriented data structure is utilized, which allows storing and describing image regions using low-level descriptors. Different types of structural relationships between regions are also taken into account. A distinctive goal of this work is that data operations are adapted for working in a distributed mode. This allows that an input image can be divided into different sub-images that can be stored and processed separately by different agents in the system, facilitating processing very-large images in a parallel manner. A key aspect to decrease processing time for parallelized tasks is the use of an appropriate load balancer to distribute and assign tasks to agents with less workload.

Keywords:
Cloud computing, image processing, load balancing, image region analysis, data structures, image storage, digital libraries, distributed databases, parallel processing, very-large datasets.

1. Introduction

A large number of gigabytes of multimedia information are being generated every day over the world. Institutions, hospitals, companies, and governments are producing large image and video collections. Methods for image storing and processing are needed for managing these multimedia collections. Particularly, the data volumes and processing difficulty associated to very-large images pose a challenging problem. This work aims to simplify and facilitate working with very-large image datasets. The final purpose is to create a cloud computing service capable of storing and analyzing very-large image datasets. The service allows this way users to access additional virtualized computing and storage resources. Particularly, the service enables to analyze very-large images by providing both computational and physical resources.

Nowadays there exist some types of images in certain application areas with very-large sizes due to the progressive increase of resolution and quality in capturing instruments. For example, the Blue Marble project of NASA (National Aeronautics and Space Administration) obtains images whose sizes can be over 1 GB (PNG file) [1]. Another example

1877-0509
doi:10.1016/j.procs.2010.04.300

can be certain high resolution medical images, such as ultra-high optical coherence tomography images. Working with these images implies large memory and processor resources. These datasets are difficult to manage, and their sizes continue to increase.

Our case of study is centered in such very-large images, which normally have an extremely large number of regions. This implies that storing and processing the image data require large computing resources, as well as time. To diminish these storing and processing requirements, database parallelization and distributed processing are used.

A cloud computing service prototype for storing and analyzing images has been developed. The service stores an image and its associated information. Regions, and their relationships, are used as the basic entities for both representing and processing an image. Therefore, the system must be able (i) to extract the image regions (and their relevant relationships), (ii) to process, and (iii) to store them. A problem is that the amount of regions in an image is usually very-large, and the relationships between them can be relatively complex.

Associated to the presented cloud computing service, a data structure for storing very-large images has been implemented. A data structure for analyzing images must permit to manage the contained information easily and quickly. In our system, the information contained in the data structure is stored in a database. Besides, it is transparent for service clients whether they are operating over a database.

Parallelization can be used for large collections of images [2]. Most often, such a parallelization simply consists in spreading images in different separated databases, which allows to increase performance by doing parallel image search simultaneously in different databases. Besides this kind of parallelization, we have developed a dividing algorithm for very-large images in which an input image is divided into a number of sub-images, which can be stored in different databases and processed (in certain operations) by different machines. After that, every sub-image can be accessed separately as an independent image or as a part of a greater image. Consequently the data structure has been adapted for distributed processing. Our multi-agent system provides a transparent uniform access to all component sub-images.

This paper is organized as follows. Section 2 describes the cloud computing service, the data structure, and the dividing algorithm and associated operations. The cloud computing service is described in section 3. Results are discussed in Section 4, and conclusions can be found in Section 5.

2. Methods

2.1. Data Structure

A basis of any image analysis system is an adequate data-structure for storing the relevant information available in an image. Structural relationships between entities appearing in an image should be taken into account. Some previous works that use region based representations are [3][4][5]. Each region is described by its physical features — color, shape, and position descriptors are used. This representation also allows defining several structural relationships between different regions in the image.

A graph-based data structure is used in our system to store the information of an input image. Each image region (in principle, a connected set of image pixels with same label value) is represented as a graph node of the data structure. And every region relationship that relates a region with other ones corresponds to a graph arc.

Table 1 shows the descriptor features that the system uses to describe regions. Nevertheless, this feature set can be enlarged and adapted to specific applications. As Table 1 shows, the data structure also considers and stores region relationships. The nature of the four relationships currently considered is quite different. The first one, the 'adjacent-to' relationship, is strictly low level, and it can be obtained directly from the input image data and connectivity. The next one, the 'is-part-of' relationship, provides information about the structure of the image, and it indicates regions that belong to other regions. The last two relationships, 'related-to' and 'disjoint-with', are normally specific to the particular image application. The 'related-to' relationship is used to relate separated regions that are not adjacent in the image. For example, it is normally used to associate separated regions that belong to the same object. The 'disjoint-with' relationship is employed to force separation between adjacent regions (e.g., for adjacent regions that are perhaps quite similar but that do belong to different objets). In addition, regions can be annotated with textual information (not shown in Table 1).

A relational database schema that contains the descriptors of regions (and their relationships) indicated in Table 1 has been developed.

Color descriptors	Shape descriptors	Relationships
red level average	area	adjacent-to
green level average	major axis size	is-part-of
blue level average	minor axis size	related-to
red level mode	major axis orientation	disjoint-with
green level mode	minor axis orientation	
blue level mode	centroid	
red level variance	set of region points	
green level variance	set of border points	
blue level variance	average of centroid-border distance	
maximum red level value	variance of centroid-border distance	
maximum green level value	number of sides estimation	
maximum blue level value		
minimum red level value		
minimum green level value		
minimum blue level value		

Table 1: Classification of descriptors obtained for an image region.

2.2. Dividing algorithm

The dividing algorithm is used to distribute parts of an image in different databases. This image division must enable distributed processing so that, for certain operations, the result of processing an entire image can be computed by merging and composing the results of processing all sub-images. Therefore, the input image must be divided following some requirements, which are related to the operation to be applied; it is not generally possible dividing an input image into sub-images and processing each one separately, and to expect to obtain an identical result to that computed by processing the input image as a whole for all types of operations. This is an important and complex topic on its own, although we will focus in this paper on the description of the cloud computing service and the increase of performance rather than on the constraints and relationships between the image division step and some graph-based operations.

In the dividing algorithm, the input image is divided along its largest dimension (height or width) to obtain three sub-images I_A, I_B and I_C by using the following algorithm:

```
FOR EACH region IN divisionLine DO
   IF (NOT isTreated(region)) THEN
      regionSet = getSimilarRegions (region)
      FOR EACH region_r IN regionSet DO
          markAsTreated(region_r)
      END FOR
      regUnion = \begin{math}\cup\end{math} regionSet
      // regUnion : union of regions in regionSet
      I_A = I_A \begin{math}\setminus\end{math} regUnion
      I_B = I_B \begin{math}\cup\end{math} regUnion
      I_C = I_C \begin{math}\setminus\end{math} regUnion
   END IF
END FOR
```

`divisionLine` is the middle line along its largest dimension (height or width). `getSimilarRegions` grows the input region with adjacent ones that are similar, until a similarity condition is not satisfied. Two regions a and b are similar if all three following expressions are satisfied:

$$\max(|r_a - r_b|, |g_a - g_b|, |b_a - b_b|) < th_{max} \tag{1}$$

$$\min(|r_a - r_b|, |g_a - g_b|, |b_a - b_b|) < th_{min} \tag{2}$$

$$\frac{|r_a - r_b| + |g_a - g_b| + |b_a - b_b|}{3} < th_{avg} \tag{3}$$

Note: (r_a, g_a, b_a) and (r_b, g_b, b_b) denote the three RGB values of regions a and b, respectively.

In this manner, three sub-images are obtained. The central part of the image (sub-image I_B) has been treated in the process of division and is ready to be inserted in a database. On the other hand, the left and right part of the input image (sub-images I_A and I_C) are unprocessed images, and they can be sent to be treated by other computers. Some additional information, part of the central sub-image I_B, is sent altogether with the I_A and I_C sub-images. This information is used for describing region neighborhoods that are in adjacent sub-images.

A messaging interface has been developed in order to synchronize information backwards between the sub-images when the entire image has been treated. Whereas, initially, the I_A and I_C sub-images have information about adjacent I_B regions, I_B needs to be informed about its outside boundary when I_A and I_C are processed and stored (in principle, in remote databases).

Finally, when the synchronization step has finished, the image is prepared to be analyzed using graph operations adapted to the parallel data structure.

2.3. Graph operations

The operations defined on the data structure are region-based. We use mainly morphological graph-based operations [6][7]. Elementary graph neighborhoods, which involve each region and its adjacent neighboring regions, are usually employed. For example, a graph dilation (or, respectively, erosion) of size 1 would compute the maximum (respectively, minimum) intensity value of each region and those of its neighboring regions.

Methods for working with the data structure directly on a database have been developed. SQL code is used to perform the region-based image operations. To apply operations and to obtain information from all the sub-images that form an image, in general different databases have to be queried [8] [9]. When a query is received, the integration engine spreads the query to the underlying databases of the distributed system. Each database processes the query and returns the corresponding result. Our integration engine is capable of merging all the sub-results received from different databases in order to create a unique result set.

3. Cloud computing service

3.1. Service implementation

The cloud computing service has been implemented as a multi-agent system. A schema of the system is depicted in Figure 1.

First of all, a distributed and parallel system have to define how tasks are assigned to an agent. In order to minimize response and execution time, the implemented system uses a load balancing technique for distributing tasks between agents. Each agent has a local task queue, but a central information agent. Benefits of using load balancing algorithms for multi-agent, parallel and grid systems have been extensively studied [10][11][12]. Following the analysis in [11] and [10], load balancing in our system has been developed exploiting benefits of maintaining a centralized resource index; at the same time, the number of network packets sent is reduced by sending tasks directly to local queues in agents. These local queues also allow rebalancing when an agent gets too busy. Load Balancing uses three principal functionalities:

(i) *Service advertisement and discovery* is used by different agents to locate where other agents can be accessed and to publish their own location.

(ii) *Performance prediction* is used to assign new tasks to a determined agent in order to equilibrate execution time for different agents. Each agent registers its system performance for each type of operation (division, storage, and morphological operations). Using task parameters (for example, number of regions or untreated pixels), execution time is predicted using a logarithmic regression function. And the task is assigned to the agent that has less estimated pending processing time.

(iii) *Queues of tasks* are employed for scheduling purposes. When a new task is assigned to an agent, maybe other tasks are being executed by this agent. Each agent queues the tasks pending execution in a local queue.

An important issue in multi-agent systems is the role assignment of the agents that exist in the system. As Figure 1 shows, there are three kinds of agents in our system:

- The *service access point agent* is a unique agent in the system. This agent is the entrance point for final users. It offers the methods for storing and applying operations of the cloud computing service.

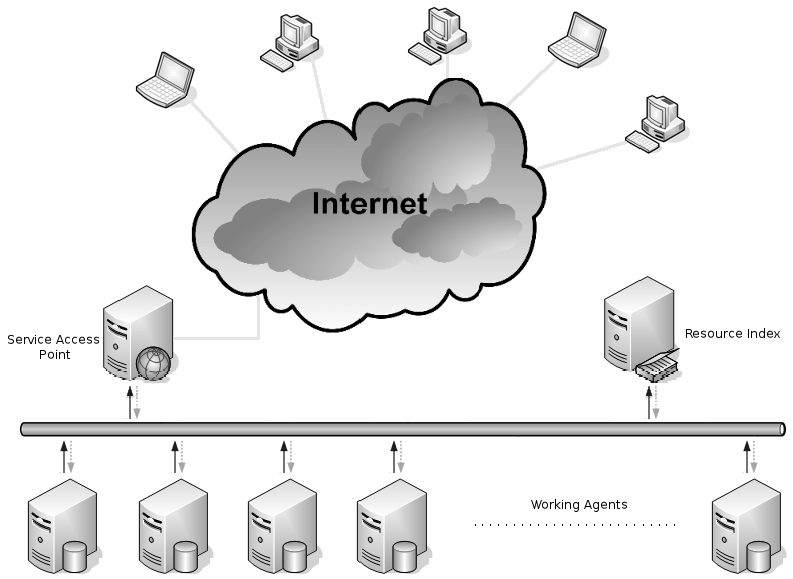


Figure 1: Cloud Computing Service Schema

- The *resource index agent* is also a unique agent in the system. It implements the service directory. This agent collects information about the working agents in the system. The collected information contains the location of each agent, its current load, the estimation function for performance prediction, and the image processing status.
- Finally, *working agents* are the main agent role in the system. Each working agent manages a relational database containing the complete data-structure commented in Section 2.1. They have all necessary functionalities for storing and analyzing images on its own. So, every working agent is capable of (a) dividing an input image (when it is too large), (b) extracting image regions and descriptors, (c) storing image information into the data-structure, and (d) applying certain filters and graph-based operations on stored images in their own database.

Additionally these agents have a messaging interface for both synchronizing the information of the divided images, and sending the sub-images obtained in the dividing algorithm that have to be treated by other agents. Working agents are capable also of examining their own system load to compute the pending execution time estimation commented before. This data acquisition is crucial for load balancing, which is commented in the next section.

The implemented multi-agent system has been developed using Sun Java. Each working agent can be executed in heterogeneous hardware, with heterogeneous database management systems and over different operating systems. The system could be easily upgraded by registrating new agents in the resource index. The number of agents in the system is not limited.

3.2. Load balancing description

In the implemented system, load balancing is primarily used for image division and storage. This is the first step for an input image. In this step the image is distributed among different agents. For this purpose agent behavior has five main steps. The agent (a) receives an image to be processed, (b) decides if the image has to be divided, (c) divides the image when necessary (using the algorithm described in 2.2) and computes its regions and descriptors, (d) stores the obtained regions and descriptors in the data-structure, and, finally, (e) sends, if the image has been divided, the two

obtained untreated sub-images to two agents (the least stressed). Step (c) requires processing all the pixels of the sub-images. Therefore, our system uses the *number of untreated pixels* of the received sub-image to estimate the pending execution time. This also allows to calculate an estimation of the agent loads, necessary in step (e) to distribute tasks among agents. The load estimation takes into account all pending tasks in the local queue of the agents.

Operations performed on different sub-images does not usually require load balancing. This is so because the operation is performed directly on the data-structure where the sub-image is stored. Operations performance is dependant on the number of regions that are assigned to each agent.

A key task of load balancing is obtaining the load of each working agent. The system load balancing is used in the dividing and storing step.

Every working agent has a main scheduler thread. This thread manages the local queue containing the pending tasks to be executed by the working agent. When the working agent receives a new incoming task from other agent, the scheduler thread adds the new task to the local queue. The number of concurrent tasks executing in an agent is configurable: the scheduler thread controls the number of tasks that are being executed by the agent at the same time, and it launches a new thread for another task from the local queue when necessary.

The scheduler stores, for each task, the start time, the number of untreated pixels, and the finish time. The scheduler agent uses those three parameters of previously executed task threads for estimating the processing time of incoming tasks. A logarithmic regression function is utilized for this issue. Every task in the agent has its own time estimation.

When the local queue changes, the scheduler thread updates the information of the working agent in the resource index agent. This way, in the resource index agent is stored the estimated time for processing the pending tasks in all working agents. So when a working agent have to send a new task (a sub-image to be processed by other working agent), it queries the resource index agent regarding which is the less utilized working agent at the moment. Agents use Web Services as messaging interface between them.

Additionally, another Java thread in working agents is the load monitoring thread. It is responsible for registering the system resources consumption per executed task thread. This monitoring thread has to register only those threads of the agent that are executing tasks. Due to how the Sun Java thread model is implemented, all Java threads in a machine are executed inside of the Java Virtual Machine process. So this load monitoring have to register the CPU usage per task thread. For this purpose Sun JMX Beans are used. By using the real CPU usage figure in addition to the total processing time of each task, we can refine the estimation function when more than one task is executed at the same time.

3.3. Balancing the number of regions

A homogeneous region data distribution among agents is important for reducing processing time for operations on images. Initially, the total number of regions in an image cannot be known. As a consequence of that, the dividing algorithm load balancing is based on untreated pixels. Therefore, the number of pixels stored in different agents is near homogeneous but the number of regions is not necessarily.

In order to increase the performance of operations, an optional step for rebalancing the number of regions in agents is used. When sub-images are inserted, the information about the destination agent and the number of regions of each sub-image is sent and stored in the resource index agent. When all sub-images have been processed, the resource index can calculate the total number of regions of the image and the desired ideal number of regions per agent. This generates a rebalancing strategy for migrating some regions to keep their number between agents balanced.

The basis for this migration strategy is using the Best Fit Descending Algorithm, which is widely used in ordering and bin packing problems. We treat agents as bins in the algorithm, and regions have an initial allocation. To avoid re-processing regions and relationships, all regions inside a sub-image are not divided. Sub-images from those agents that surpass the target region number are assigned to other agents that have less regions.

4. Results and discussion

We will present in this section some experimental results of the performance increase in the implemented service by distributing the image division and storage tasks for several input images. Benefits of using multiple agents with

Image id	Image size	Piecewise-constant regions	Relations between regions	Size in MB
Image 1	1000x1000	1,961	6,904	3.81
Image 2	1067x1749	4,095	12,752	5.33
Image 3	1232x1507	42,421	248,794	5.31
Image 4	1596x2178	21,451	150,078	9.94
Image 5	5000x5000	67,723	367,022	95.3
Image 6	16601x9201	9,037,812	48,818,032	610.3

Table 2: Test images description

load balancing, instead of using just one agent (and, therefore, not distributing processing), are shown. Afterwards, a graph-based morphological dilation operation is applied to show the benefits of parallelizing operations as well.

Six test images have been selected from the Blue Marble project by NASA, which are described in Table 2. They have different sizes (ranging from 1000×1000 to 16601×9201 pixels) and different number of piecewise-constant regions for testing the service performance for a range of image sizes. As can be observed in images 3 and 4, the number of initial regions is not directly dependent on the number of pixels of the image (an image could be very-large while at the same time its number of piecewise-constant regions being relatively small).

In order to better study the system load balancing and performance, homogeneous machines and operating systems have been used. Agents have been installed in desktop computers with Intel QuadCore processor with 4 GB of RAM and a 64-bits Linux operating system.

Each test image is inserted into our image analysis system, which divides it into several sub-images. The latter ones are processed separately. The performance increase of the division and storage step when using different number of agents is depicted in Figure 2. Processing time for an agent is represented by a percentage, where the 100% mark denotes the processing time needed for completing the task using only one agent. For cases using more than one agent, the processing time for an operation is the maximum processing time of all agents involved.

As shown in Figure 2(a), the time for diving and storing an image in the data-structure using two agents is in average about one half (49,78 %) of the total time for storing using only one agent. This time is reduced to 35.50 % using three agents (see Figure 2(b)). We can say that the load balancing for storing images is quite effective. This is so because the time that is needed for extracting the initial regions and storing an entire image is proportional to the number of pixels of the image. The number of pixels is known *a priori*. Afterwards, tasks can be balanced among different agents thanks to the estimation functions mentioned in Section 3. On the other hand, two sub-images having the same number of pixels could have a very different number of regions. In this case, the required time for the storing step is greater in sub-images with more regions and relationships. This occurs, for example, in image 3 as can be seen in Table 3.

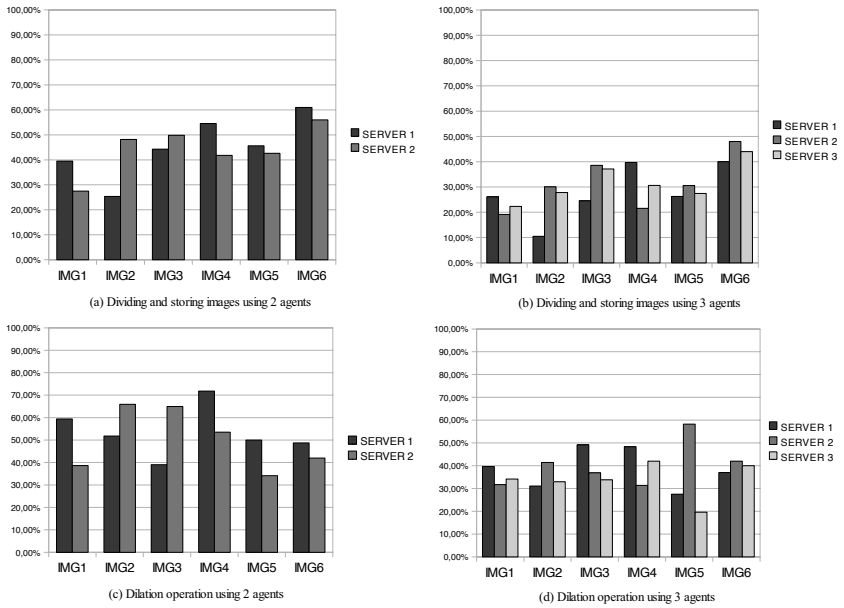
A morphological graph-based dilation is applied separately to each sub-image. Figure 2 (c) and (d) shows the processing time needed to apply dilation to the whole image using two and three agents compared to using a single agent, which are in average 60.14% and 46.46%, respectively. The considered operations in the system are region oriented. Thus, the region number distribution between different agents (see Table 3) is the factor that mainly determines the time processing reduction.

As mentioned before, not all operations can be distributedly computed as in this example. Some operations in general cannot, whereas for others the dividing algorithm should be adapted.

5. Conclusion and future work

This paper has presented a cloud computing service for image analysis that focuses on the challenges and problems posed by very-large images. It has been implemented using (a) a region-based data structure for storing and processing very-large image data in distributed databases, and (b) a distributed cooperative multi-agent system.

Adequate parallelization and workload balancing of our distributed system are crucial features to ensure an improved performance. The operations for managing and processing the data-structure can work directly on databases.



(a) Dividing and storing images using 2 agents

(b) Dividing and storing images using 3 agents

(c) Dilation operation using 2 agents

(d) Dilation operation using 3 agents

Figure 2: Time comparison with different number of agents

In the implemented multi-agent system each working agent can be executed in heterogeneous hardware, with heterogeneous database management systems and over different operating systems. In future research, the system is intended to be extended with other morphological distributed graph operations as well as other more complex operations, such as a graph-based watershed.

6. Acknowledgments

This work has been supported in part by "Ministerio de Ciencia e Innovación" of Spain (Ref.: TIN2007-61768).

Image id	One Agent	Two Agents	Three Agents
Image 1	[1,961]	[1,159; 802]	[872; 676; 413]
Image 2	[4,095]	[1,999; 2,196]	[1,202; 1,518; 1,375]
Image 3	[42,421]	[19,869; 22,552]	[16,949; 13,457; 12,015]
Image 4	[21,451]	[12,828; 8,623]	[8,630; 5,084; 7,737]
Image 5	[67,723]	[38,682; 29,041]	[20,121; 31,920; 15,682]
Image 6	[9,037,812]	[4,881,432; 4,156,380]	[2,884,214; 3,167,826; 2,985,772]

Table 3: Piecewise-constant regions division in different agents

7. References

[1] R. E. Wolfe, D. P. Roy, E. Vermote, MODIS land data storage, gridding, and compositing methodology: Level 2 grid, IEEE Transactions on Geoscience and Remote Sensing 36 (1998) 1324–1338. doi:10.1109/36.701082.

[2] D. Picard, M. Cord, A. Revel, CBIR in distributed databases using a multi-agent system, in: IEEE International Conference on Image Processing (ICIP'06), 2006.
URL `http://publi-etis.ensea.fr/2006/PCR06`

[3] B. Marcotegui, Segmentation de sequences d'images en vue du codage, Ph. D. Thesis, École Nationale Supérieure de Mines de Paris, 1996.

[4] P. Salembier, L. Garrido, Binary partition tree as an efficient representation for image processing, segmentation, and information retrieval, IEEE Transactions in image processing 9 (4) (2000) 561–576.

[5] W. Yu, J. Fritts, F. Sun, A hierarchical image segmetation algorithm, in: IEEE International Conference on Multimedia and Expo, 2002.

[6] J. Serra (Ed.), Mathematical Morphology. Volume I and Volume II: theoretical advances, London: Academic Press, 1988.

[7] P. Soille, Morphological Image Analysis, 2nd Edition, Springer-Verlag, Heidelberg, 2003.

[8] D. Pérez-Rey, V. Maojo, M. García-Remesal, R. Alonso-Calvo, H. Billhardt, F. Martín, A. Sousa, Ontofusion: Ontology based integration of genomic and clinical databases, Computers in Biology and Medicine 2006 7-8 (36) (2006) 712–730.

[9] R. Alonso-Calvo, V. Maojo, M. García-Remesal, F. Martín, H. Billhardt, D. Pérez-Rey, An agent and ontology-based system for integrating public gene, protein, and disease databases, Journal of Biomedical Informatics 40 (2007) 17–29.

[10] W. Leinberger, G. Karypis, V. Kumar, R. Biswas, Load balancing across near-homogeneous multi-resource servers, in: Heterogeneous Computing Workshop, 2000. (HCW 2000) Proceedings. 9th, 2000, pp. 60–71.

[11] S. A. J. Junwei Cao, Daniel P. Spooner, G. R. Nudd, Grid load balancing using intelligent agents, Future Generation Computer Systems 21 (1) (2005) 135–149.

[12] B. Yagoubi, Y. Slimani, Task load balancing strategy for grid computing, Journal of Computer Sciences 3 (3) (2007) 186–194.